ECE 375 LAB 4

Data Manipulation and LCD Display

Lab Time: Friday 4-6

Faaiq Waqar

Jordan Brown

Introduction

The purpose of this lab was to practices data manipulation and using the LCD display. We began by learning how to initialize the stack, registers used during the program, and configuring peripherals such as the LCD. The LCD uses many different library functions and we learned to use as well as learning how to include the driver file in our project. The main challenge of this lab is managing the memory and moving it from one place to another. Because of the difference in size between pointers in data memory and program memory other methods must be used to move between locations. To accomplish this process students learn to use functions in a way to mimic a while loop, which can move the contents of the string into a different part of memory. Once the string has arrived in data memory it can be read out to the LCD screen.

Program Overview

The majority of the program happens in the WHILE, NEXT, and NICE functions. These are responsible for moving over the strings into data memory so the characters can be used in the LCD library functions. The function “WHILE” loops until the first string has been completely moved. This process is mirrored by another set of functions that complete the same task. Between these two sets is a function called “NEXT” that prepares the next string to be moved. It serves a similar purpose to several lines found in the INIT stage of the code that will begin the process of starting the secondary string to be carried from program memory into the data memory, that we will be able to access and display onto the LCD screen afterwards. Note that in the section we will take on bit at a time, starting at the section of the data memory referenced to as $0110 or $0100, and then record the 16 bits, using a program post increment for the addressing register during each iteration to make use of the data in the form of an array.

Next we get into the main portion of the code. Here, we will continue the program into what can be compared to an infinite while loop, one of the likes we saw when developing our C bump bot program with a set condition to 1. We want to be checking for 3 primary instances, built into a sort of state machine that the assembly code will move into on the basis of button presses that will eb compared to on the basis of a read from PIND, the determiner for read on the buttons on the AVR board. In the instances of both the pin press on one and the pin presses on two, we will enter a portion of the function that makes use of the LCD Write on One and Two that will take the strings that we wrote to program memory in the first section of the doc, and translate that into output on the screen. However, we have established a register for bit orientation that is register 19. This will be used to establish whether the orientation needs to be switched. This is to eliminate the perspective of unusually large amounts of orientation switches being caused by any one button hit. In the case of the the button 7 push, we will use the LCD clr function on both lines to clear out the screen and data, then call to INIT to make sure we have available memory used for the next instruction.

Finally, we can discuss how the byte switch function works. As with almost any swap function seen there are 2 sections of memory, but in this case 2 registers, that will take the values, and place them in the opposite positions of sting one and string 2, then operate a loop until all the bits have been swapped. This is a function that will be called to a lot, making it a great choice to use in this form, rather than hardcoding it, as it makes the process faster as a whole.

Init

The initialization runs once right when the power is turned on, and setups up the stack pointer and registers that will be used. We declare our strings that are used when printing to the board and also initialize the LCD. The strings

are loaded from program memory into the Y register. Their definitions are then moved to the low and high bytes of Z

WHILE/NICE

This loop moves each character into data memory by moving it to register r16 and then by moving that value into data memory. A check is done to see if we have reached the address indicating the end of the string. If this condition is met then we will enter another function, and if not met then the loop will continue. “Nice” is the function responsible for performing the check and jumping to “Next” if the process is complete.

NEXT

“Next” repeats several lines that we executed during the initialization. We have given Y the address from other string and the process described above will be repeated. The next string will be moved into data memory 1 character at a time by using a register between the 2 locations.

MAIN

The main function comprises of the primarily loop in which data will be both received from the PIND buttons, and the functions used in order to execute the specific operations for the display, reverse display, and clearing of information to the LCD screen that make part as the primary portion of the program. This includes operations for the button presses on 1 and 0 and 7, all operated on indicated binaries.

ADDITIONAL QUESTIONS

Study Questions 1. In this lab, you were required to move data between two memory types: program memory and data memory. Explain the intended uses and key differences between these two memory types.

Program memory is where the individual assembly instructions are stored when we flash the board. Data memory is where our general purpose registers, I/O registers, variables, and intermediate results. The avr command LPM can be used to use program memory within data memory, in the case of lab 4.

2. You also learned how to make function calls. Explain how making a function call works (including its connection to the stack), and explain why a RET instruction must be used to return from a function.

When subroutines are called the stack is used to store return addresses and necessary values. These are pushed onto the stack which allows the code to trace back through the previous lines. The stack pointer always points 1 positions above the top of the stack and it grows towards the end of memory. IIn our lab 4 code we initialize the end of the stack to eb the beginning of the sram. These values are popped off the stack when we return to the subroutine and the stack can be incremented to get the next values.

3. To help you understand why the stack pointer is important, comment out the stack pointer initialization at the beginning of your program, and then try running the program on your mega128 board and also in the simulator. What behavior do you observe when the stack pointer is never initialized? In detail, explain what happens (or no longer happens) and why it happens.

If we are to not initialize the stack pointer, the behavior of program changes when we attempt to use function calls that rely on the usage of some subroutine, as an example the flip function used that we have implemented to flip

bytes when the instructed button is pressed. The reason for this is that without its initialization, the stack is not usable in program flash, and the stack is incredibly important because it is the source that allows us to store the spot we would like to return to in subroutine calls, as the address of the PC+1 spot is stored onto the stack in the RJMP subroutine calls. Without a usable stack pointer, we will end up not being able to store this, making it so we can jump to a subroutine call, but never to return again.

Difficulties

Our only difficulties came from referencing functions that were not named properly, but this was quickly resolved. Taking inputs for each of the buttons also took some practice and we referred to the lab 1 source code to check that we had gotten the I/O process correct. We also were able to repurpose the code used to delay the movement of the letters on the LCD when we were working on the challenge code.

Conclusion

This lab was challenging, but a lot of very important information was covered. The concepts covered are critical for success in the class and getting experience seeing how they work was helpful. Also seeing a good template for how the majority of assembly files will be coded will be a nice example to refer to during future labs. This introduction into function and loops also showed us good examples of how to declare and call use them. With branch statements we can compare variables or the results of calculations to determine how many times to loop or when we need to call another function. These capabilities allow us to reuse code and make things simpler to read.

Source Code

```
;***********************************************************

;*

;*  Faaiq_Waqar_and_Jordan_Brown_Sourcecode.asm

;*

;*      This program will display the contents of two

;*      Strings from PM and shift them into the LCD display,

;*      With manipulations done with button 0,1 & 7 Pushes

;*

;*      Source Code file for Lab 4 of ECE 375

;*

;***********************************************************

;*

;*       Author: Faaiq Waqar and Jordan Brown

;*         Date: Enter date
```

```
;*
;***********************************************************

.include "m128def.inc"                    ; Include definition file

;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                         ; Multipurpose register is
                                                  ; required for LCD Driver

;***********************************************************
;*      Start of Code Segment
;***********************************************************
.cseg                                             ; Beginning of code segment

;***********************************************************
;*      Interrupt Vectors
;***********************************************************
.org    $0000                             ; Beginning of IVs
               rjmp INIT                          ; Reset interrupt

.org    $0046                             ; End of Interrupt Vectors

;***********************************************************
;*      Program Initialization
;***********************************************************
INIT:                                             ; The initialization routine
               ; Initialize Stack Pointer
               ldi r16, low(RAMEND)  ; Prepare lower stack addr
```

```
        out SPL, r16                ; Store lower stack addr
        ldi r17, high(RAMEND) ; Prepare upper stack addr
        out SPH, r17                ; Store upper stack addr
        ; Initialize LCD Display
        rcall LCDInit               ; Routine call LCD initialize
        ; Move strings from Program Memory to Data Memory
        ldi YL, low($0100)          ; Load immediate value of $0100
                                            ; In memory to address reg Y (low)
        ldi YH, high($0100)         ; Load immediate value of $0100
                                            ; In memory to address reg Y (high)
        ; Move the location of the string definition in PM to Z (low)
        ldi ZL, low(STRING_BEG<<1)
        ; Move the location of the string definition in PM to Z (high)
        ldi ZH, high(STRING_BEG<<1)
        ldi r19, 0                      ; Initialize r19, used for reversal
                                            ; Indicator flag

;*********************************************************
;*      While Loop for String One Initialization
;*********************************************************
WHILE:
        lpm r16, Z+                     ; Load contents of first string
                                            ; Char into the register, post-inc
        st Y+, r16                      ; Store contents of register into
                                            ; Data memory, and post-inc
        ; Compare the current location of the Z pointer to string-end
        cpi ZL, low(STRING_END<<1)
        breq NICE                       ; If at the end of string (highadd)
                                            ; Go to the next check
        rjmp WHILE                      ; Loop of not EQ
```

```
NICE:
              ; Check the conents of the higher bits of Z addr
              cpi ZH, high(STRING_END<<1)
              breq NEXT                         ; If EQ, branch to the next section
              rjmp WHILE                        ; Otherwise loop again
;***********************************************************
;*     String Two Initialization (Data Memory)
;***********************************************************
NEXT:
              ldi YL, low($0110)           ; Start the process again, prepare
              ldi YH, high($0110)          ; Register Y sections with addr $0110
              ldi ZL, low(STRING_BEGJ<<1) ; Once again, set up Z, this time
              ldi ZH, high(STRING_BEGJ<<1) ; With the Start of the next string
;***********************************************************
;*     While Loop for String Two Initialization
;***********************************************************
WHILEB:
              lpm r16, Z+                       ; Store contents of Z in r16, post-inc
              st Y+, r16                        ; Store contents of r16 in Y, post-inc
              ; Compare Z-Low bits to check for end of address string
              cpi ZL, low(STRING_ENDJ<<1)
              breq NICEB                        ; If equivalent, branch to next portion
              rjmp WHILEB                       ; Loop if not equal
NICEB:
              ; Check bits for high inputs on Z (eos)
              cpi ZH, high(STRING_ENDJ<<1)
              breq NEXTB                        ; If equivalent, end of string, end
              rjmp WHILEB                       ; Loop if not equivalent
NEXTB:
              ; NOTE that there is no RET or RJMP from INIT, this
```

```
              ; is because the next instruction executed is the
              ; first instruction of the main program


;***********************************************************
;*     Main Program
;***********************************************************
MAIN:
              in r20, PIND                ; Take input for Buttons, store in R20
              cpi r20, 0b11111110         ; Check for Button 0 Input
              breq BUTTONPD0              ; if pressed, branch to function pd0
              cpi r20, 0b11111101         ; Check for Button 1 Input
              breq BUTTONPD1              ; If pressed, branch to function pd1
              cpi r20, 0b10111111         ; Check for Button 7 Input
              breq BUTTONPD7              ; If pressed, branch to function pd7
              rjmp MAIN                         ; If nothing is pressed, continued loop
BUTTONPD0:
              cpi r19, 1                        ; Check if the swap register is set to rev
              breq PD0JMP                       ; If it is set to rev, send to rcallpd0
              rjmp PD0WRT                       ; Otherwise, jump to display
PD0JMP:
              rcall FUNC                        ; Call to reversal of bits function
PD0WRT:
              ldi r19, 0                        ; Make sure status of swap re is set forw
              rcall LCDWrLn1              ; Print Line 1 to LCD
              rcall LCDWrLn2              ; Print Line 2 to LCD
              rjmp MAIN                         ; Loop back to the main
BUTTONPD1:
              cpi r19, 0                        ; Check if the swap reg is set to forw
              breq PD1JMP                       ; if it is, prepare for func jump
              rjmp PD1WRT                       ; Otherwise, send to display portion
```

```
PD1JMP:
                rcall FUNC                          ; Call to swapping bits function
PD1WRT:
                ldi r19, 1                          ; Make sure status of swap reg is set to
rev
                rcall LCDWrLn1              ; Print Line 1 to LCD
                rcall LCDWrLn2              ; print Line 2 to LCD
                rjmp MAIN                           ; Jump back to main loop
BUTTONPD7:
                rcall LCDClrLn1                     ; Clear the lines in LCD 1
                rcall LCDClrLn2                     ; Cleat the lines in LCD 2
                rjmp INIT                           ; Jump to INIT to have data ready for new
call
                ; The Main program
                ; Display the strings on the LCD Display
                ; jump back to main and create an infinite
                                                    ; while loop.  Generally, every main
program is an
                                                    ; infinite while loop, never let the
main program
                                                    ; just run off


;***********************************************************
;*      Functions and Subroutines
;***********************************************************


;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-----------------------------------------------------------
FUNC:                                               ; Begin a function with a label
```

```
                ; Save variables by pushing them to the stack
                ldi YL, low($0100)          ; Prepare the Y addr with high
                ldi YH, high($0100)         ; And low bits for $0100
                ldi XL, low($0110)          ; Prepare the X addr with high
                ldi XH, high($0110)         ; And low bits for $0110
                ldi r21, 0                          ; Set counter to r21 for 0
                ldi r22, 1                          ; Set inc to r22 for 1
WHILEC:
                ld r17, Y                           ; Load R17 with Y memory
                ld r18, X                           ; Load R18 with X memory
                st Y+, r18                          ; Post-Inc and store X mem to Y
                st X+, r17                          ; Post-Inc and store Y mem to X
                add r21, r22                ; Increment the counting reg
                cpi r21, 12                         ; Compare the counter to max
                breq ENDFUNC                ; If equivalent, end of loop
                rjmp WHILEC                         ; If not, loop again
                ; Execute the function here

                ; Restore variables by popping them from the stack,
                ; in reverse order
ENDFUNC:
                ret                                         ; End a function with RET


;***********************************************************
;*      Stored Program Data
;***********************************************************

;-----------------------------------------------------------
; An example of storing a string. Note the labels before and
; after the .DB directive; these can help to access the data
```

```
;-----------------------------------------------------------

STRING_BEG:

.DB            "Faaiq Waqar "        ; Declaring data in ProgMem

STRING_END:


STRING_BEGJ:                         ; Declaring String 2 in ProgMem

.DB            "Jordan Brown"

STRING_ENDJ:


;***********************************************************

;*      Additional Program Includes

;***********************************************************

.include "LCDDriver.asm"             ; Include the LCD Driver
```

CHALLENGE CODE

```
;***********************************************************

;*

;*  Faaiq_Waqar_and_Jordan_Brown_Challengecode.asm

;*

;*      This program will display the contents of two

;*      Strings from PM and shift them into the LCD display,

;*      With manipulations done with button 0,1 & 7 Pushes

;*  This is the challenge with dynamic display

;*

;*      Source Code file for Lab 4 of ECE 375

;*

;***********************************************************

;*

;*       Author: Faaiq Waqar and Jordan Brown
```

```
;*         Date: October 2019
;*
;***********************************************************

.include "m128def.inc"                    ; Include definition file

;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                         ; Multipurpose register is
                                                  ; required for LCD Driver

;***********************************************************
;*      Start of Code Segment
;***********************************************************
.cseg                                             ; Beginning of code segment

;***********************************************************
;*      Interrupt Vectors
;***********************************************************
.org    $0000                             ; Beginning of IVs
               rjmp INIT                          ; Reset interrupt

.org    $0046                             ; End of Interrupt Vectors

;***********************************************************
;*      Program Initialization
;***********************************************************
INIT:                                             ; The initialization routine
               ; Initialize Stack Pointer
```

```
                ldi r16, low(RAMEND)  ; Prepare lower stack addr
                out SPL, r16                ; Store lower stack addr
                ldi r17, high(RAMEND) ; Prepare upper stack addr
                out SPH, r17                ; Store upper stack addr
                ; Initialize LCD Display
                rcall LCDInit               ; Routine call LCD initialize
                ; Move strings from Program Memory to Data Memory
                ldi YL, low($0100)          ; Load immediate value of $0100
                                                    ; In memory to address reg Y (low)
                ldi YH, high($0100)         ; Load immediate value of $0100
                                                    ; In memory to address reg Y (high)
                ; Move the location of the string definition in PM to Z (low)
                ldi ZL, low(STRING_BEG<<1)
                ; Move the location of the string definition in PM to Z (high)
                ldi ZH, high(STRING_BEG<<1)
                ldi r25, 0                          ; Initialize r19, used for reversal
                                                    ; Indicator flag

;*********************************************************
;*      While Loop for String One Initialization
;*********************************************************
WHILE:
                lpm r16, Z+                         ; Load contents of first string
                                                    ; Char into the register, post-inc
                st Y+, r16                          ; Store contents of register into
                                                    ; Data memory, and post-inc
                ; Compare the current location of the Z pointer to string-end
                cpi ZL, low(STRING_END<<1)
                breq NICE                           ; If at the end of string (highadd)
                                                    ; Go to the next check
```

```
                rjmp WHILE                          ; Loop of not EQ
NICE:
                ; Check the conents of the higher bits of Z addr
                cpi ZH, high(STRING_END<<1)
                breq NEXT                           ; If EQ, branch to the next section
                rjmp WHILE                          ; Otherwise loop again
;***********************************************************
;*      String Two Initialization (Data Memory)
;***********************************************************
NEXT:
                ldi YL, low($0110)           ; Start the process again, prepare
                ldi YH, high($0110)          ; Register Y sections with addr $0110
                ldi ZL, low(STRING_BEGJ<<1) ; Once again, set up Z, this time
                ldi ZH, high(STRING_BEGJ<<1) ; With the Start of the next string
;***********************************************************
;*      While Loop for String Two Initialization
;***********************************************************
WHILEB:
                lpm r16, Z+                         ; Store contents of Z in r16, post-inc
                st Y+, r16                          ; Store contents of r16 in Y, post-inc
                ; Compare Z-Low bits to check for end of address string
                cpi ZL, low(STRING_ENDJ<<1)
                breq NICEB                          ; If equivalent, branch to next portion
                rjmp WHILEB                         ; Loop if not equal
NICEB:
                ; Check bits for high inputs on Z (eos)
                cpi ZH, high(STRING_ENDJ<<1)
                breq NEXTB                          ; If equivalent, end of string, end
                rjmp WHILEB                         ; Loop if not equivalent
NEXTB:
```

```
                ; NOTE that there is no RET or RJMP from INIT, this
                ; is because the next instruction executed is the
                ; first instruction of the main program

;***********************************************************
;*      Main Program
;***********************************************************
MAIN:
                in r20, PIND                ; Take input for Buttons, store in R20
                cpi r20, 0b11111110         ; Check for Button 0 Input
                breq BUTTONPD0              ; if pressed, branch to function pd0
                cpi r20, 0b11111101         ; Check for Button 1 Input
                breq BUTTONPD1              ; If pressed, branch to function pd1
                cpi r20, 0b10111111         ; Check for Button 7 Input
                breq BUTTONPD7              ; If pressed, branch to function pd7
                rjmp MAIN                           ; If nothing is pressed, continued loop
BUTTONPD0:
                cpi r25, 1                          ; Check if the swap register is set to rev
                breq PD0JMP                         ; If it is set to rev, send to rcallpd0
                rjmp PD0WRT                         ; Otherwise, jump to display
PD0JMP:
                rcall FUNC                          ; Call to reversal of bits function
PD0WRT:
                ldi r25, 0                          ; Make sure status of swap re is set forw
                rcall LCDWrLn1              ; Print Line 1 to LCD
                rcall LCDWrLn2              ; Print Line 2 to LCD
                rcall WAITFUNC
                rcall SHFTFUNC              ; Call to shift for challenge portion
                in r20, PIND                ; Take input for Buttons, store in R20
                cpi r20, 0b11111101         ; Check for Button 1 Input
```

```
                breq BUTTONPD1              ; If pressed, branch to function pd1
                cpi r20, 0b10111111         ; Check for Button 7 Input
                breq BUTTONPD7              ; If pressed, branch to function pd7
                rjmp PD0WRT                         ; Loop back to the main
BUTTONPD1:
                cpi r25, 0                          ; Check if the swap reg is set to forw
                breq PD1JMP                         ; if it is, prepare for func jump
                rjmp PD1WRT                         ; Otherwise, send to display portion
PD1JMP:
                rcall FUNC                          ; Call to swapping bits function
PD1WRT:
                ldi r25, 1                          ; Make sure status of swap reg is set to
rev
                rcall LCDWrLn1              ; Print Line 1 to LCD
                rcall LCDWrLn2              ; print Line 2 to LCD
                rcall WAITFUNC
                rcall SHFTFUNC              ; Call to shift for challenge portion
                in r20, PIND                ; Take input for Buttons, store in R20
                cpi r20, 0b11111110         ; Check for Button 1 Input
                breq BUTTONPD0              ; If pressed, branch to function pd1
                cpi r20, 0b10111111         ; Check for Button 7 Input
                breq BUTTONPD7              ; If pressed, branch to function pd7
                rjmp PD1WRT                         ; Jump back to main loop
BUTTONPD7:
                rcall LCDClrLn1                     ; Clear the lines in LCD 1
                rcall LCDClrLn2                     ; Cleat the lines in LCD 2
                rjmp INIT                           ; Jump to INIT to have data ready for new
call
                ; The Main program
                ; Display the strings on the LCD Display
                ; jump back to main and create an infinite
```

```
                                                        ; while loop.  Generally, every main
program is an
                                                        ; infinite while loop, never let the
main program
                                                        ; just run off


;***********************************************************
;*      Functions and Subroutines
;***********************************************************


;-----------------------------------------------------------
; Func: FUNC Reversal Function
; Desc: Reverse the strings in data mem based on button push
;-----------------------------------------------------------
FUNC:                                           ; Begin a function with a label
               ; Save variables by pushing them to the stack
               ldi YL, low($0100)           ; Prepare the Y addr with high
               ldi YH, high($0100)          ; And low bits for $0100
               ldi XL, low($0110)           ; Prepare the X addr with high
               ldi XH, high($0110)          ; And low bits for $0110
               ldi r21, 0                           ; Set counter to r21 for 0
               ldi r22, 1                           ; Set inc to r22 for 1
WHILEC:
               ld r17, Y                            ; Load R17 with Y memory
               ld r18, X                            ; Load R18 with X memory
               st Y+, r18                           ; Post-Inc and store X mem to Y
               st X+, r17                           ; Post-Inc and store Y mem to X
               add r21, r22                 ; Increment the counting reg
               cpi r21, 16                          ; Compare the counter to max
               breq ENDFUNC                 ; If equivalent, end of loop
               rjmp WHILEC                          ; If not, loop again
```

```
                ; Execute the function here


                ; Restore variables by popping them from the stack,
                ; in reverse order
ENDFUNC:
                ret                                     ; End a function with RET


;-----------------------------------------------------------
; Func: SHFTFUNC Shifting Function
; Desc: Used for bonus portion, shift bits 1+ in datamem
;-----------------------------------------------------------
SHFTFUNC:
                ldi YL, low($010F)          ; Prepare the Y addr with high
                ldi YH, high($010F)         ; And low bits for $010F
                ldi r22, 0                          ; Load the counter to prep iter
                ld r23, Y                           ; Load the contents of string end to reg
                ldi YL, low($0101)          ; Prepare the y addr with high
                ldi YH, high($0101)         ; And low bits for $0101
                ldi XL, low($0100)          ; Prepare the X addr with high
                ldi XH, high($0100)         ; And low bits for $0100
WHILESHFT:
                ld r24, X                           ; Load the contents of X into a temp
                st X+, r23                          ; Store the prev value to x, post inc
                mov r23, r24                ; Move contents of this register into the
                                                    ; Setting one to make room for next
val
                ld r24, Y+                          ; Load the value at Y addr into temp reg
                inc r22                             ; increment Counter
                cpi r22, 15                         ; Check counter comparison
                breq ENDSHFT                ; End the shifter if the counter finished
```

```
                rjmp WHILESHFT              ; Continue loop if not equal
ENDSHFT:
                st X+, r23                          ; Shift the final bit
SHFTFUNCB:
                ldi YL, low($011F)          ; Prepare the Y addr with high
                ldi YH, high($011F)         ; And low bits for $011F
                ldi r22, 0                          ; Prepare counter
                ld r23, Y                           ; Load memory of end of string
                ldi YL, low($0111)          ; Prepare the Y addr with high
                ldi YH, high($0111)         ; And low bits for $0111
                ldi XL, low($0110)          ; Prepare the X addr with high
                ldi XH, high($0110)         ; And low bits for $0110
WHILESHFTB:
                ld r24, X                           ; Load x data mem into temp reg
                st X+, r23                          ; Store the prev value to x, post inc
                mov r23, r24                ; Move data from r24 to r23
                ld r24, Y+                          ; Load addr mem into r24 from Y, post inc
                inc r22                             ; Increment Counter
                cpi r22, 15                         ; Check counter for max hit
                breq ENDSHFTB               ; Branch out of loop if complete
                rjmp WHILESHFTB                     ; Loop if not equivalent
ENDSHFTB:
                st X+, r23                          ; Complete the shift into last datamem
                ret                                         ; Return using stack addr


; *********************************************************
;*  Wait CALL Function (Using Reg, 18, 19,20)
; *********************************************************
WAITFUNC:
                push    r18                 ; Save wait register
```

```
                push    r19                     ; Save ilcnt register
                push    r20                     ; Save olcnt register

                ldi             r18, 15
Loop:   ldi             r20, 224                ; load olcnt register
OLoop:  ldi             r19, 237                ; load ilcnt register
ILoop:  dec             r19                     ; decrement ilcnt
                brne    ILoop                   ; Continue Inner Loop
                dec             r20             ; decrement olcnt
                brne    OLoop                   ; Continue Outer Loop
                dec             r18             ; Decrement wait
                brne    Loop                    ; Continue Wait loop

                pop             r20             ; Restore olcnt register
                pop             r19             ; Restore ilcnt register
                pop             r18             ; Restore wait register
                ret                             ; Return from subroutine
;***********************************************************
;*      Stored Program Data
;***********************************************************

;-----------------------------------------------------------
; An example of storing a string. Note the labels before and
; after the .DB directive; these can help to access the data
;-----------------------------------------------------------
STRING_BEG:
.DB             "Faaiq Waqar     "              ; Declaring data in ProgMem
STRING_END:

STRING_BEGJ:                            ; Declaring String 2 in ProgMem
```

```
.DB            "Jordan Brown    "
STRING_ENDJ:


;*****************************************************
;*      Additional Program Includes
;*****************************************************
.include "LCDDriver.asm"            ; Include the LCD Driver
```